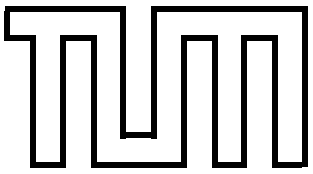


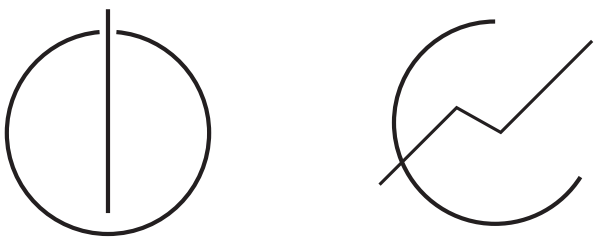
FAKULTÄT FÜR INFORMATIK
und
FAKULTÄT FÜR
WIRTSCHAFTSWISSENSCHAFTEN

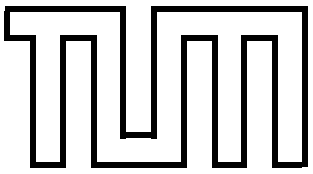
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Technische Dokumentation

Flowgame

Philipp Reichart, Barbara Köhler,
Christopher Hübner und Sebastian Vöst





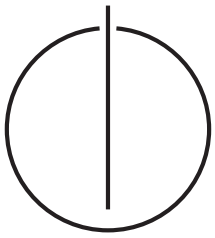
FAKULTÄT FÜR INFORMATIK
und
FAKULTÄT FÜR WIRTSCHAFTSWISSENSCHAFTEN

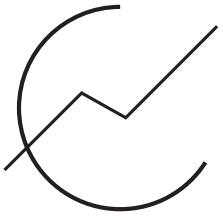
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Technische Dokumentation

Flowgame

Autoren:	Philipp Reichart, Barbara Köhler, Christopher Hübner und Sebastian Vöst
Aufgabensteller:	Prof. Dr. phil. Hugo M. Kehr
Betreuer:	Dipl. Psych. Stefan Engeser
Datum:	4. März 2011





Inhaltsverzeichnis

1 Funktionale Anforderungen

Der Zustand des „Flow“ ist ein psychologisches Phänomen, das bei diversen verschiedenen Tätigkeiten zu beobachten ist. Computerspiele zählen zu diesen Tätigkeiten, da sie die grundlegenden Anforderungen zur Generierung von Flow erfüllen: Der Spieler muss aktiv sein und einer gewissen Herausforderung gegenüberstehen.
In diesem Projekt soll also zunächst einmal ein Spiel entwickelt werden, das den Spieler potentiell in einen Zustand des Flow bringt, oder, so gewollt, in einen Zustand des „Nicht- Flow“.

Als Spielkonzept wurde ein simples Setting gewählt: Der Spieler steuert ein Raumschiff und muss dabei Hindernissen ausweichen und Boni aufsammeln, die mit variabler Geschwindigkeit auf ihn zukommen. Anhand der Geschwindigkeit lässt sich hierbei der Schwierigkeitsgrad regulieren, um so gezielt Unterforderung, Überforderung oder ideale Anforderungen schaffen zu können.

Zur Navigation im Spiel ist es nötig, zunächst ein Hauptmenü zu integrieren, in dem man wählen kann, ob man eine Runde beginnen, die Highscores einsehen, die persönlichen Einstellungen ändern oder die Credits ansehen möchte.
Bei neuen Spielern wird zunächst ein Persönlichkeitsprofil angelegt, in dem der Spieler mithilfe eines Fragebogens gewisse persönliche Charakteristika angibt, die zur Auswertung der erhobenen Daten hilfreich sind.
Ist dies geschehen, kann ein Spiel beginnen. Dieses ist aufgeteilt in mehrere Runden, in denen zunächst die „Tagesform“, also die Leistungsfähigkeit für diese Runde gemessen wird. Dies ermöglicht, trotz Lerneffekten oder anderen Einflüssen, wie etwa Müdigkeit oder mangelnde Konzentration, in jeder Situation die idealen Anforderungen zu generieren.
Daraufhin folgen drei Runden, in denen sich verschiedene Anforderungsstufen abwechseln. Um die Rahmenbedingungen für alle Spieler gleich zu halten, hat jede Runde eine fixe Dauer.

Das ganze Spiel ist, um etwas Atmosphäre zu erzeugen, in eine Rahmengeschichte eingebettet, die den Spieler in ein Science-Fiction-Szenario hineinversetzt und ihm vor jeder Runde eine Aufgabe gibt, die er zu erfüllen hat und ihm am Ende der Runde über seine Erfolge berichtet.
Zusätzlich wird das Befinden in einer Art „Rennen“ durch ein schnelles, futuristisch erscheinendes Musikstück unterstützt, das während den Runden abgespielt wird und speziell hierfür komponiert wurde.

Um anhand hiervon nun festzustellen, ob ein Spieler Flow erfährt, muss eine Methode gewählt werden, um dies zu messen.
Das Mittel der Wahl sind hier Teile der Fragebögen der von Engeser entwickelten FKS[1], die zwischen den Spielrunden eingeblendet werden, sowie weitere Fragen zur Anforderungspassung, um das subjektive Empfinden der angepassten Schwierigkeit überprüfen

[1] TODO: Erklärung

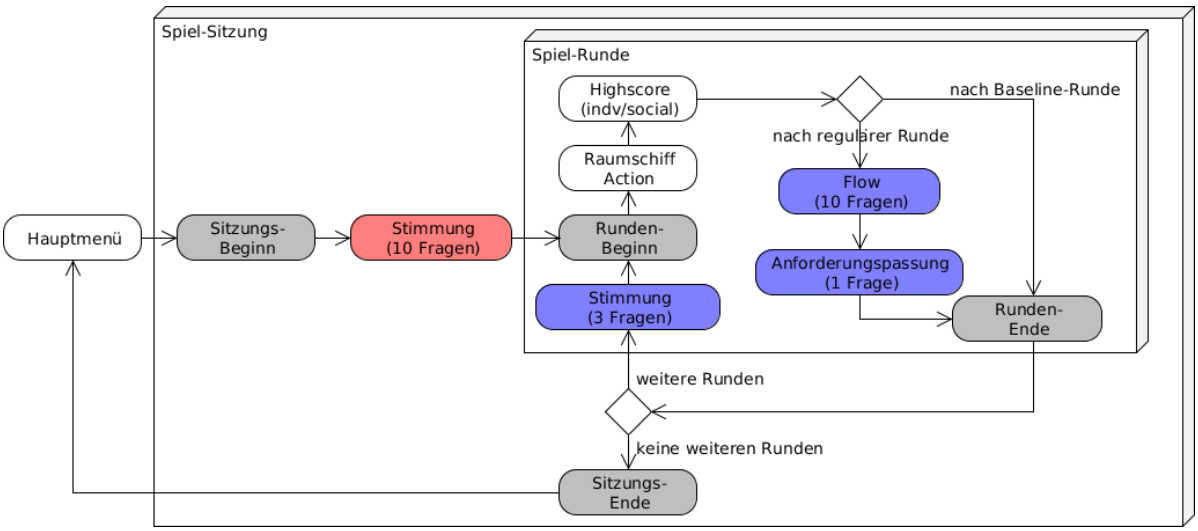


Abbildung 1: Clientseitiger Spielablauf

zu können. Die Beantwortung dieser Fragen erfolgt durch einen stufenlos anpassbaren Schieberegler, der zwischen zwei gegensätzlichen Adjektiven nach Belieben eingestellt werden kann. Ein Lerneffekt beim Beantworten der Fragen oder eine Automatisierung ebendessen wird verhindert, indem die Fragen in randomisierter Reihenfolge erscheinen. Außerdem wird die Dauer der Beantwortung eines jeden Fragebogens gemessen, was Rückschlüsse darauf zulässt, ob der Spieler nur zufällig etwas angeklickt, oder die Fragen tatsächlich gelesen und gewissenhaft beantwortet hat.

Als Plattform für das Spiel wurde das soziale Netzwerk Facebook[2] gewählt, das die Integration von vielfältigen Anwendungen erlaubt. Mit Facebook lassen sich vielerlei Daten, die der Spieler auf seinem Profil ohnehin angegeben hat, wie etwa Alter, Geschlecht und Herkunft, nutzen. Diese Daten können dann automatisiert und ohne zusätzliche Angaben erfasst werden und in die Auswertung einfließen.
Außerdem ist es hier möglich,die Leistungen verschiedener Teilnehmer zu vergleichen und in einer Highscore darzustellen. Das erlaubt die Untersuchung von Unterschieden der Leistung von individuell und sozial motivierten Spielern.

Zuletzt wurde eine einfache und umfassende Möglichkeit der erhobenen Daten gefordert. Dies ließ sich durch eine direkte Einspeisung in eine SQL[3]-Datenbank realisieren. Dies hat den Vorteil, dass die Daten elektronisch und delokalisiert vorliegen und ein Zugriff über das Internet möglich ist. Hierfür steht ein browserbasierter Client zur Verfügung. Zudem lässt sich die Datenbank im SPSS-Format[4] exportieren.

[2] `http://www.facebook.com`
[3] Structured query language
[4] Datenformat der Statistiksoftware SPSS Statistics.

2 Nicht-Funktionale Anforderungen

Um gleiche Rahmenbedingungen und damit eine Vergleichbarkeit zwischen verschiedenen Spielern und Spielen nicht von vornherein zu verhindern, muss der Spielablauf in eine möglichst kontrollierte Umgebung eingebettet werden. Das bedeutet, dass sich verschiedene Spiele nicht zu stark voneinander unterscheiden dürfen. Ein kritischer Punkt hierfür ist die Dauer einer Runde. Es steht zu befürchten, dass Spieler trotz gleichem Schwierigkeitsgrad signifikant unterschiedliche Erlebnisse berichten, wenn sich die Spieldauer in großem Maße unterscheidet. Daher wurde für jede Runde eine fixe Spieldauer von 120 Sekunden festgelegt. Diese Dauer lässt sich auch nicht durch ein „verlieren“ oder „sterben“, wie ich anderen Spielen üblich verkürzen. Man muss in diesem Spiel zwar Hindernissen ausweichen, diese führen aber nie zum Spieltod, sondern immer nur zu einer Verringerung der Endpunktzahl. Diese Endpunktzahl übt als Maß zur Leistungsmessung für den Spieler einen Anreiz für hohe Performance und damit Anstrengung aus. Daher ist es sinnig, dass der Spieler neben dem zwanghaften Ausweichen von Hindernissen auch noch eine Anforderung darin hat, gleichzeitig möglichst viele Bonuspunkte zu sammeln.

Obwohl es sich hierbei um ein psychologisches Experiment handelt, ist es wünschenswert, dass der Spieler nicht das Gefühl hat, in einem Labor an einer Datenerhebung teilzunehmen, sondern vielmehr ein Spiel spielt, um sich die Zeit zu vertreiben oder ähnliches. Daraus resultieren die oben genannten Anforderungen, das Spiel mit Rahmengeschichte und Musik auszustatten, da dies von einem Spiel erwartet wird und ein Fehlen solcher Details zu einem verfälschten Empfinden führen könnte. Ebenso gehört dazu, dass die Fragebögen möglichst gut ins Spiel eingebettet werden und ihre Beantwortung durch eine Rahmengeschichte eingepasst wird. Müssten die Fragebögen extern bearbeitet werden, etwa per Papier oder durch einen weiteren Webclient, so wäre es unvermeidlich, dass der Spieler sich mehr als Proband empfindet.

Ziel des Projekts ist außerdem, möglichst viele Datensätze zu erheben, also möglichst viele Menschen zu erreichen. Dafür muss das Spiel allerdings einige Anforderungen erfüllen und nicht zu hohe Hürden an den Benutzer stellen. Das bedeutet zunächst eine möglichst einfache Bedienbarkeit. Angestrebt waren lediglich vier Tasten, nämlich die Pfeiltasten, die ausreichend sind, das Raumschiff nach oben, unten, links und rechts zu steuern. Ebenso dazu gehört es, das Spiel möglichst kurzweilig zu gestalten. Ein Spiel (bestehend ja aus vier Runden) sollte weniger als 15 Minuten dauern, um der kurzlebigen Internetkultur zu entsprechen. Im Internet wird oft kurz „zwischendurch“ oder beim Warten gespielt und eine längere Dauer würde das Spiel hierfür komplett ausschließen. Außerdem ist es wichtig, den Anforderungen der verschiedenen Betriebssysteme, die verbreitetsten sind Windows, Mac/OSX und Linux, gerecht zu werden. Teile der potentiellen Teilnehmerschaft wegen ihres Betriebssystems ausschließen zu müssen, wäre äußerst unerfreulich. Ebenso gibt es keinen Grund, das Projekt auf den deutschen Raum zu beschränken. Eine Sprachunterstützung von Englisch und Deutsch soll die Anzahl der potentiellen Teilnehmer noch einmal signifikant erhöhen. Dazu wurden die Standards

der in der Informatik so genannten „I18N“ (kurz für Internationalisierung) eingehalten. Teil dessen ist, dass Programmteile, die sich bei unterschiedlichen Sprachversionen unterscheiden, nicht im Quellcode stehen, sondern einfach durch den Austausch von bestimmten Dateien ersetzt werden können. Dies ermöglicht die Erweiterung auf andere Sprachräume, ohne das eigentliche Programm zu ändern.

3 Architektur

3.1 Client-Server

Die grundlegende Architektur des Spiels ist Client-Server-basiert, wobei ein auf Facebook angezeigtes Applet den Client und ein auf einem eigenen Server laufender Webcontainer den Server darstellen. Die Kommunikation zwischen Client und Server erfolgt ausschließlich über HTTP[5]. Programmcode für Funktionalitäten, die spezifisch für Client (Benutzerschnittstelle, 3D-Umgebung, Facebook-Anbindung) und Server (Datenbank-Anbindung, Programmlogik) sind, ist voneinander unabhängig, lediglich der Programmcode für das gemeinsam genutzte Datenmodell ist beiden Seiten bekannt. Dies entspricht der üblichen Praxis, reduziert unerwünschte Seiteneffekte und ergibt eine klare Trennung der Verantwortlichkeiten.

3.2 Subsystemzerlegung

TODO Christopher, Phil

3.3 Datenmodell (Klassen / UML)

TODO Christopher, Phil

3.4 Datenbankmodell (Tabellen / ER)

TODO Christopher, Phil

[5]Hypertext Transfer Protocol – Protokoll zur Übertragung von Daten über ein Netzwerk, hauptsächlich für die Übermittlung von Webseiten eingesetzt.

4 Technologien

Um den Entwicklungsaufwand für ein plattformübergreifendes Spiel möglichst gering zu halten, haben wir uns entschieden, dass Spiel in Java[6] umzusetzen, da es für alle verbreiteten Betriebssysteme verfügbar ist und eine solide Bibliothek an Funktionen mit sich bringt.

Serverseitig läuft eine Webanwendung auf Basis von Struts 2[7] und Servlets/JSPs, die über EclipseLink als JPA[8]-Provider mit einer Datenbank kommuniziert, um Spieldaten zu speichern. Für die Kommunikation mit dem clientseitigen Spiel wird ein auf serialisierte Java-Objekte aufbauender RPC[9]-Mechanismus über HTTP angewandt.

Clientseitig läuft ein Applet[10], dass über Java3D die 3D-Umgebung und über Swing[11] die Benutzerschnittstelle anzeigt. Beide Bibliotheken abstrahieren die zugrundelegenden Techniken stark, was eine komfortable Entwicklung komplexer 3D-Umgebungen und Benutzerschnittstellen ermöglicht. Für die Anbindung an Facebook wird im Applet eine weitere Bibliothek verwendet, die aufgrund von Sicherheitseinschränkungen (Same Origin Policy) alle Aufrufe an das Facebook API über den Spielserver abwickelt.

5 Implementierung

5.1 Baselinemessung und Adaptive Rundenzeiten

Für die Erzeugung und Messung von Flow bzw. Unter- und Überforderung passt sich das Spiel dem Spieler an. Dazu findet bei jedem Durchgang, einer sogenannten Session, zunächst eine Messung der Baseline statt. Diese Baseline stellt die vermutete ideale Anforderung an einen Spieler dar. Alle weiteren Runden werden im Verhältnis zur Baseline im Schwierigkeitsgrad variiert.

In diesem Kapitel wird zunächst kurz die Messung der Baseline erläutert und Schwierigkeiten die dabei auftreten können. Anschließend wird gezeigt, wie die Geschwindigkeit der übrigen Runden aus der Baseline errechnet werden kann. Baselinemessung

Bei der Baselinemessung soll erreicht werden, dass man die ideale Spielschwierigkeit für einen Spieler ermittelt. Diese besteht prinzipiell aus den drei Komponenten Geschwin-

[6] Java ist eine moderne, statisch typisierte, objektorientierte Programmiersprache. Kompilierte Java-Programme benötigen eine Laufzeitumgebung (Virtuelle Maschine, auch Java Runtime genannt), die für viele unterschiedliche Betriebssysteme zur Verfügung steht und dadurch einen hohen Grad an Plattformunabhängigkeit sicherstellt.

[7] Java Framework zur Entwicklung von Webapplikationen – `http://struts.apache.org`

[8] Java Persistence API, eine Schnittstelle um Java-Objekte-Graphen in gebräuchliche relationale Datenbanksysteme abzubilden.

[9] Remote Procedure Call, das Aufrufen von Programmcode auf einem anderen Computersystem.

[10] Im Webbrowser laufendes Java-Programm

[11] Bibliothek zur Grafikausgabe unter Java

digkeit, Verhältnis von Diamanten zu Asteroiden und Abstand zwischen den Asteroiden. Der Einfachheit halber wird in unserem Experiment ausschließlich die Geschwindigkeit variiert, eine Anpassungen der beiden anderen Parameter könnte aber analog erfolgen.

Die Passung soll ermittelt werden, indem zunächst alle auf der gleichen Geschwindigkeit starten. Nach und nach wird die Geschwindigkeit erhöht, bis der Spieler nicht mehr mithalten kann. An diesem Punkt wird wieder abgesenkt, bis es anscheinend deutlich zu einfach wird. Dies wird in mehreren Iterationen über zwei Minuten hinweg wiederholt, wobei der Ausschlag nach und nach kleiner wird und sich auf einer bestimmten Schwierigkeit einpendelt. Die Baseline ist dann die Schwierigkeit oberhalb derer sich der Spieler 50% der Zeit halten konnte. Dieses Verhalten führt zu dem in Abbildung 2 dargestellten typischen Verlauf.

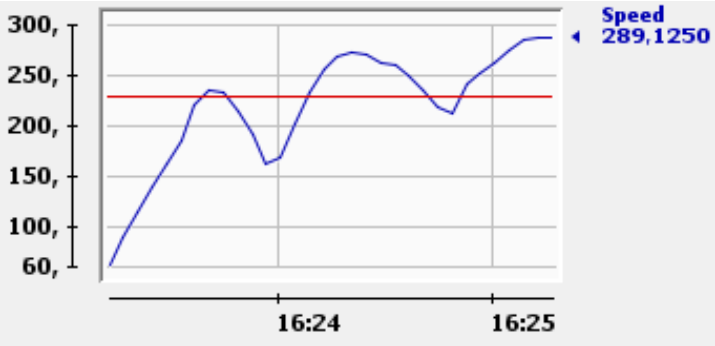


Abbildung 2: Baselinemessung Verlauf (blaue Linie), gemessene Baseline (rote Linie)

Als erstes muss also zunächst bestimmt werden, ob die Schwierigkeit zu einem gegebenen Moment zu groß oder zu klein ist. Dazu speichert die Klasse `Trend`, zu wieviel Prozent der Spieler mit einem bestimmten Objekt (Diamant oder Asteroid) kollidiert ist bzw. ihm ausgewichen ist. Es wird jeweils ein lang-, mittel- und kurzfristiger Trend ermittelt, sodass die Entscheidung nicht nur anhand der letzten Aktionen des Spelers getroffen werden. Der langfristige Trend bezieht sich hierbei auf die letzten 30 passierenden Objekte. Mittelfristig sind es die letzten 10 und für den kurzfristigen Trend werden nur die letzten 3 beachtet.

Beispiel: Es sind 10 Objekte in der folgenden Reihenfolge vorbeigeflogen, wobei ein (K) bedeutet, dass der Spieler mit ihnen kollidiert ist:

1. Asteroid
2. Asteroid
3. Diamant (K)
4. Asteroid
5. Diamant (K)
6. Diamant

7. Diamant (K)
8. Asteroid (K)
9. Asteroid (K)
10. Diamant

Dann ist der kurzfristige Trend, dass der Spieler mit 2 Asteroiden kollidiert ist (Rate 100%) und keinen Diamanten gesammelt hat (0%), was eine Überforderung vermuten lässt. Der mittelfristige Trend ist mit 2 Asteroiden (40%) und 4 Diamanten (80%) dagegen deutlich ausgewogener.

Aus dieser Information muss nun abgeschätzt werden, wann das Spiel langsamer und wann es schneller werden sollte. Hierfür wurden mehrere verschiedene Methoden getestet. Im Folgenden wird lediglich die beschrieben, die tatsächlich verwendet wurde, da sie sich als am exaktesten herausgestellt hat.

Bis der Spieler mindestens zehn Objekte passiert hat wird die Geschwindigkeit einfach immer gesteigert, da man bis zu diesem Moment noch nicht genug Daten hat, um eine individualisierte Entscheidung zu treffen. Danach richtet sich die Anpassung der Geschwindigkeit nach den gemessenen Trends. Kollidiert der Spieler mit einem Asteroiden oder schafft er es nur 1/3 der Asteroiden aufzusammmeln so wird kurz abgebremst. Sammelt er dagegen erfolgreich mehr als 2/3 der Diamanten, so wird sehr stark beschleunigt. Sammelt er mittelfristig zumindest noch mehr als 1/3 der passierenden Diamanten so wird leicht beschleunigt. Wenn keine dieser Bedingungen zutrifft, so bleibt alles gleich schnell. Dieses Verhalten wird von der FunctionStrategy2 gesteuert.

Nun bleibt die Frage, wie stark beschleunigt oder gebremst werden soll. Am einfachsten erscheint es unabhängig von der aktuellen Geschwindigkeit, diese um einen fixen Betrag zu erhöhen oder zu reduzieren. Als dies mit Spielern getestet wurde haben wir jedoch festgestelt, dass bei höherer Ausgangsgeschwindigkeit schon kleine Beschleunigungen sich sehr drastisch anfühlen, während bei niedrigeren Geschwindigkeiten die Geschwindigkeitserhöhung größer sein muss, um überhaupt wahrgenommen zu werden.

Um möglichst einfach mehrere verschiedene Modelle testen zu können wird deshalb der Betrag der Erhöhung aus einer Funktion ermittelt. Die Funktion bildet eine Position auf eine Geschwindigkeit ab. Zu Beginn einer Baselinerunde startet der Spieler an Position 0. Soll das Spiel schneller werden, so wird die Position erhöht, anders gesenkt. Die neue Geschwindigkeit ergibt sich aus dem Wert der Funktion an der neuen Position.

Am besten hat sich die Funktion $\max(240 \tanh(\frac{x}{5000}) + 60, 30)$ erwiesen. Sie ist monoton steigend und bewegt sich zwischen dem niedrigsten Wert (30) und nähert sich langsam an 300. Bei dieser Funktion sind die Anpassungen bei langsamen Geschwindigkeiten deutlicher sind als bei hohen Geschwindigkeiten.

Jede Änderung der Geschwindigkeit wird im SpeedChangeBehavior angestossen. Bei der Baselinemessung merkt sich das Spiel jeden Wert und bestimmt zum Schluss die Baseline aus dem Median dieser Werte.

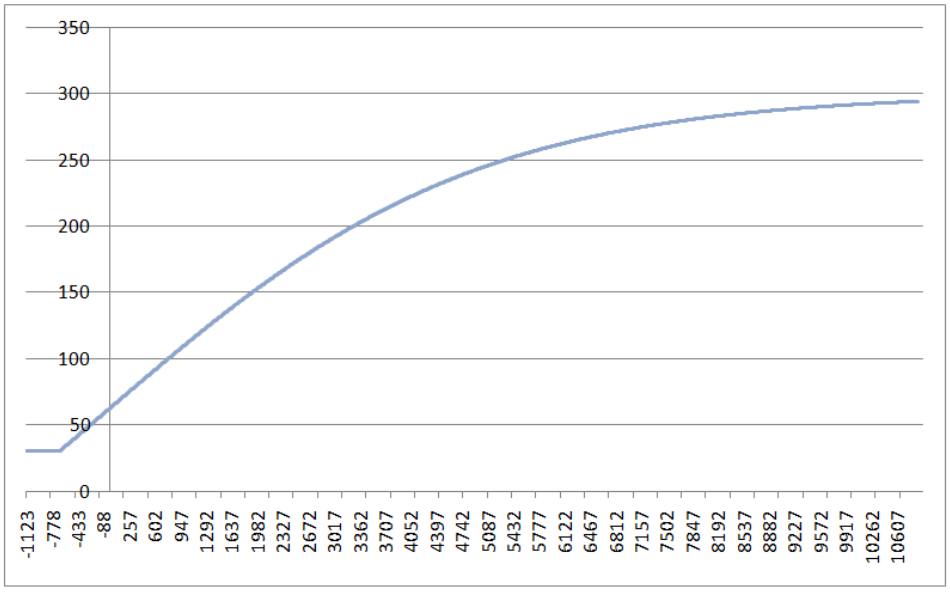


Abbildung 3: Funktion die als Grundlage für die Baselinemessung verwendet wird. TODO Vektorgrafik

Ausgehend von der Baseline werden die Geschwindigkeitenfür die folgenden drei Runden folgendermaßen bestimmt:
Stufe 1 (Baseline - 60 + Baseline*0,7)/2
Stufe 2 (Baseline - 0 + Baseline*1)/2
Stufe 3 (Baseline + 60 + Baseline*1,13)/2

Im Quellcode wurden diese wiederum durch Funktionen dargestellt. Bei der Constant-Function bleibt die Geschwindigkeit über eine Runde hinweg konstant. Die Anfangsgeschwindigkeit wird einmalig anhand der Baseline errechnet. Für die kontinuierliche Bedingung wird eine passende LinearFunction benötigt. Hierbei muss sich nicht nur der Schnittpunkt mit der y-Achse ändern sondern auch die Steigung, damit kein Sprung in der Geschwindigkeit zwischen den drei Runden besteht. Also wird zunächst die Steigung passend zur Baseline ermittelt. Daraufhin muss noch die Geschwindigkeit zum Beginn der Runde ermittelt werden. Diese liegt so, dass die mittlere Geschwindigkeit identisch wäre, zu der bei einer Runde mit konstanter Geschwindigkeit (vergleiche Abbildung 4).

5.2 Menüsystem im Spiel

Zur schnellen Entwicklung der Benutzerschnittstellen wurde auf den vollständigen Eigenbau eines Menüsystems verzichtet und stattdessen eine dünne Kompatibilitätsschicht eingeführt, die es ermöglicht, die bestehende Swing-Bibliothek von Java für die Benutzerschnittstelle des Spiels über die 3D-Umgebung zu legen. Dadurch kann die Benut-

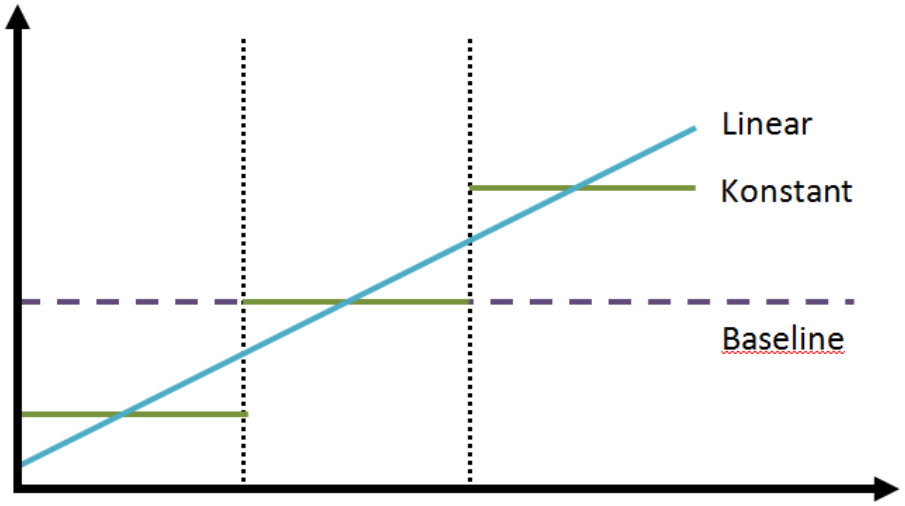


Abbildung 4: Zeigt Relation zwischen Baseline konstanter und linearer Funktion. Die konstante Funktion hat innerhalb einer Runde (gestrichelte Linie) stets die gleiche mittlere Geschwindigkeit wie die lineare Funktion.
TODO Vektorgrafik

zerschnittstelle wie gewohnt entwickelt und separat getestet werden, ohne auf die besonderen Herausforderungen Rücksicht nehmen zu müssen, die es mit sich bringt, wenn man grafische Elemente wie Buttons, Radio-/Checkboxen und Scrollleisten in einer 3D-Umgebung verwenden möchte.

Die hierfür entwickelte Kompatibilitätsschicht (`OffscreenJPanel`) zeichnet alle Swing-Komponenten in ein transparentes Bild, dass dann über die 3D-Umgebung des Spiels gelegt wird. Alle Maus- und Tastatureingaben auf diesem Bild werden abgefangen und an die eigentlichen Swing-Komponenten weitergeleitet. Während der Spieler das Raumschiff steuert, wird das komplette Swing-System ausgeblendet, was die Performance des Spiels verbessert.

Um den zugrundeliegenden Ablauf des Spiels intuitiv programmieren zu können, wurde aufbauend auf die o.g. Kompatibilitätsschicht ein Menüsystem (`GameMenu`, `MenuScreen` und das Package[12] `ui.screen`) entwickelt, dass die von Spielen typische Abfolge von „Bildschirmen“ erleichtert, zwischen denen der Spieler sich auf vordefinierten Pfaden bewegen kann.

[12]Ein Package ist eine Sammlung zusammengehöriger Java Klassen und Intefaces und ordnet den Programmcode sowohl thematisch als auch hierarchisch. Ein Package bildet per Konvention einen eindeutigen Namensraum für diese Klassen, so dass eine Unterscheidung zu Klassen gleichen Namens (in anderen Packages) möglich wird.

5.3 Persistenz von Benutzereinstellungen

Das Spiel bietet die Möglichkeit, die Sounds an- und abzustellen sowie die Steuerung zu invertieren, so dass die Auf/Ab-Pfeile sich wahlweise wie bei einem Flugzeugsimulator oder Rennspiel verhalten. Um diese Einstellungen persistent zu halten, verwendet das Applet die LiveConnect-Funktionalität moderner Webbrowser, mit denen es Javascript-Code im das Applet umgebenden HTML-Dokument aufrufen kann. Dort werden die Einstellungen dann als Cookie abgelegt und beim wiederholten Starten des Spiels automatisch ausgelesen.

5.4 RPC-Eigenbau Serialisierung über HTTP mit Java Objekt-Serialisierung

Für die Kommunikation des Spielapplets mit dem Spielserver wurde ein RPC-Mechanismus entwickelt, der sowohl einfach zu verwenden ist als auch problemlos über das Internet funktioniert. Als Grundlage dient die HTTP-Verbindung zum Spielserver, über die bereits das Spiel selbst bestehend aus Programmcode, Grafiken, Sounds, usw. geladen wird. Will das Spiel nun im Ablauf weitere Daten vom Spielserver empfangen oder an diesen senden, werden diese als Java-Objekte übergeben und serialisiert. Dies ergibt einen binären Blob[13], der als Dateiupload an den Server übertragen wird, der diesen Blob dort wieder deserialisiert um das ursprüngliche Java-Objekt zu erhalten. Abhängig von der auf dem Spielserver aufgerufenen URL, wird dieses Java-Objekt auf dem Server einem bestimmten Teil der Programmlogik übergeben, die dann damit arbeitet. Eventuelle Rückgabewerte der Programmlogik werden nach dem gleichen Verfahren durch Serialisierung von Objekten in binäre Blobs umgewandelt, als Dateidownload dem Spielapplet übermittelt und dort wieder in ein Java-Objekt deserialisiert. Aus Sicherheitsgründen werden auf dem Server auftretende Fehler nicht an das Spiel übergeben und nur in ein Logfile auf dem Server geschrieben; so wird das versehentliche Anzeigen bspw. von Passwörtern oder der Datenbankstruktur auf der Clientseite vermieden.

5.5 Java3D

Flowgame ist eine Java3D[14]-Anwendung, die als Java-Applet im Browser des Spielers (also auf dem Client) läuft. Das Universum bei Flowgame enthält vier unterschiedliche Typen von dreidimensionalen Objekten. Einen Tunnel, durch den das Raumschiff des Spielers fliegt, wobei es den entgegenkommenden Asteroiden ausweichen und Diamanten aufsammeln soll.

Die folgende Beschreibung bezieht sich auf den Stand von Revision 866. Alle Klassen der 3D-Engine befinden sich im Package `client.engine`.

[13] Binary large object
[14] Einführung unter `http://java.sun.com/developer/onlineTraining/java3d/`

Während dem Start des Applets wird in `GameApplet.java` (Zeile 79) der Szenengraph für das Spiel erstellt. Dort wird ein `Game3D` Objekt erzeugt, dass die einzelnen 3D-Objekte erstellt und in einen Szenengraph einhängt. In `Game3D.java` wird in Zeile 151 zunächst das Universum selbst instanziiert. Der Methodenaufruf in Zeile 162 erstellt die restlichen Objekte und bindet sie ein. Der Tunnel wird dabei (in `Tunnel.java`) mittels Javacode als zylinderförmige `Shape3D` erzeugt, während die anderen Objekte aus externen Objektmodelldateien eingelesen werden. Diese Objektmodelldateien befinden sich im Ressourcen Ordner (`src/res/`).

Schiffsbewegung, Steuerung und Kollisionserkennung wird mit Behaviors realisiert. Behaviors dienen bei Java3D als Basis für Interaktion und Animation. Viele dieser Aktionen müssen in jedem Frame[15] ausgeführt oder aufgerufen werden. Daher gibt es eine abstrakte `RepeatingBehavior` Klasse, die beim Neuzeichnen jedes Frames aufgerufen wird und den Zeitabstand zwischen den Frames misst. Letzteres ermöglicht eine Steuerung der Geschwindigkeit unabhängig von der Framerate. Gleichzeitig implementiert diese abstrakte Klasse das Interface `GameListener`, mit dem auf Ereignisse im Spielablauf reagiert werden kann.

`ShipNavigationBehavior` dient zur Bewegung des Schiffes in xy-Richtung, also nach links/ rechts und oben/unten. Die z-Achse entspricht bei Java3D der Tiefeninformation, Bewegung in dieser Richtung wird mit der `ForwardBehavior` realisiert, wobei die Geschwindigkeit der Bewegung wiederum von der `SpeedChangeBehavior` gesteuert wird. Über diese Behavior wird der Schwierigkeitsgrad reguliert. Die Kamera folgt dem Schiff dabei in festem Abstand, es sieht also so aus, als ob die Kollisionsobjekte (Diamanten, Asteroiden) auf den Spieler zukommen. Die `CreateCollidablesBehavior` stellt sicher, dass vor dem Schiff immer wieder neue Kollisionsobjekte erzeugt werden.

Da die bei Java3D eingebauten Möglichkeiten zur Kollisionserkennung vor allem bei höheren Bewegungsgeschwindigkeiten des Schiffes versagen, wurde auch eine eigene (einfache) Kollisionserkennung in Form der `CollisionBehavior` implementiert. Der (unendlich lange) Tunnel bewegt sich selbst nicht, die Illusion der Bewegung wird über eine Texturtransformation gesteuert. Die `TextureTransformBehavior` sorgt dafür, dass die Geschwindigkeit der Textur, zu der des Schiffes passt.

Die Behaviors werden bei der Erzeugung des 3D-Universums (s.o.) an die jeweilig betroffenen Objekte gebunden.

[15] Animierte Sequenzen in einer dreidimensionalen Umgebung werden von der Grafikhardware Bild für Bild berechnet. Eines dieser Einzelbilder wird Frame genannt. Je leistungsfähiger die vorhandene (Grafik)Hardware ist, desto mehr Bilder können pro Sekunde berechnet werden. Zur flüssigen Darstellung von Bewegung sollten stets mindestens 30 Frames/Sekunde berechnet werden können.

6 Entwicklungsumgebung

Die Entwicklungsumgebung beinhaltet alle Programme, die zum entwerfen, programmieren, kompilieren und testweisen Ausführen des Spiels und der Verwaltung des Entwicklungsprozesses nötig sind.

Für eine zentrale Verwaltung des gesamten Codes und aller weiteren Bestandteile des Spiels (Grafiken, Musik, Sounds) wurde auf Google Code ein Projekt angelegt[16], dass neben dem Versionskontrollsystem Subversion auch ein Issue-Tracking für das Verfolgen und Verwalten von Fehlern und Änderungswünschen bietet.

Als IDE[17] wird Eclipse[18] in der EE[19]-Variante mit diversen Erweiterungen verwendet: Subclipse[20] stellt die Anbindung an das von Google Code zur Verfügung gestellten Versionkontrollsystem bereit, FindBugs[21] und PMD[22] analysieren den Programmcode auf häufig auftretende Programmierfehler.
Um das Spiel vollständig lokal auszuführen, wird ein lokal installierter MySQL[23]-Datenbankserver notwendig. So können Änderungen ohne Beeinflussung des auf Facebook laufenden Spiels getestet werden.

Um jederzeit eine lauffähige Version des neuesten Entwicklungsstands zu haben, wird Hudson als CI[24]-Server verwendet, der zum einen die o.g. Werkzeuge FindBugs und PMD nochmals zentral ausführt und dazu Statistiken erzeugt und zum anderen das Deployment des Spiels auf den Facebook bekannten Spielserver vollständig automatisiert und reproduzierbar durchführen kann. Insgesamt verbessert der CI-Server damit die Code- Qualität und reduziert etwaige Fehlhandlungen beim Deployment neuer Versionen des Spiels, wodurch weniger Fehler im Spiel selbst auftreten.

7 Herausforderungen bei der Umsetzung

- Plattformübergreifende Laufzeit
  - Java Runtime + Plugin (insb. MacOSX)
  - Java3D
  - Tastatur-Events auf versch. OS

[16] `http://code.google.com/p/flowgame`
[17] Integrated Development Environment – integrierte Entwicklungsumgebung
[18] `http://www.eclipse.org`
[19] Enterprise Edition, ermöglicht unter anderem das Entwickeln von Webanwendungen
[20] `http://subclipse.tigris.org/`
[21] statisches Quellcode Analysewerkzeug – `http://findbugs.sourceforge.net`
[22] statisches Quellcode Analysewerkzeug – `http://pmd.sourceforge.net`
[23] `http://mysql.com`
[24] Continuous Integration, das durchgängige Kompilieren und Auswerten von Programmcode `http://hudson-ci.org`

- Facebook API-Änderungen (Einladungen, App-Einbindung)

- Projektmanagement
  - zu spät mit Issue Tracking angefangen
  - Zeitbudget/Auslastung Teammitglieder während Studium
  - unscharfe Anforderungen wurden zu spät festgezurrt

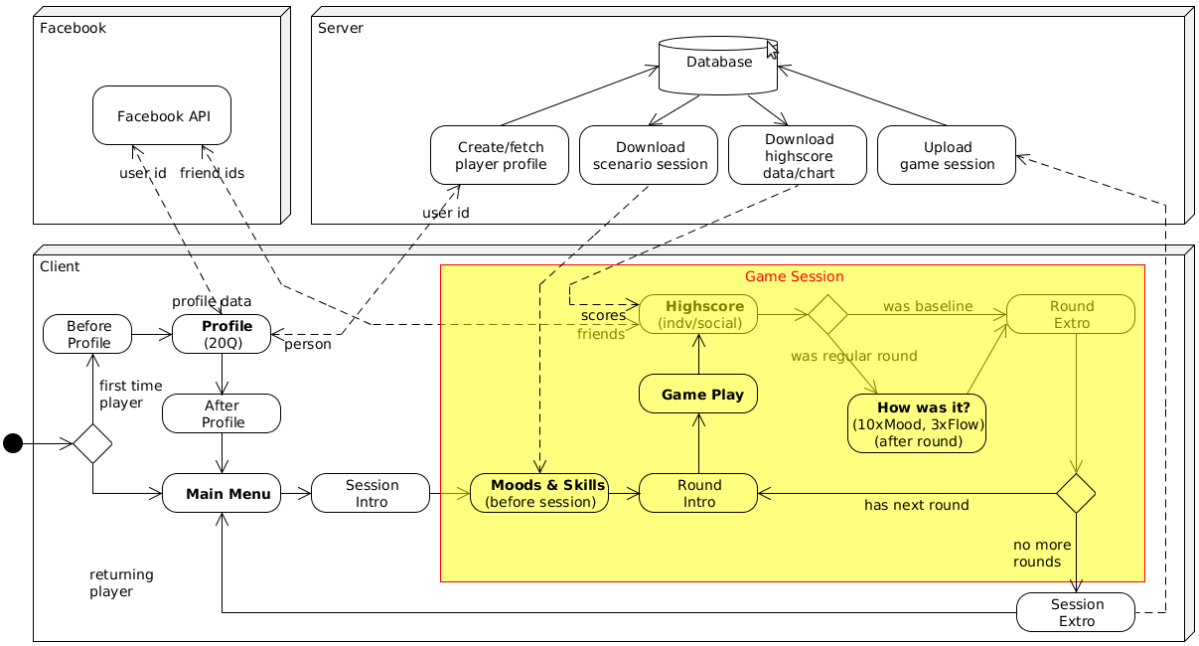


Abbildung 5: Gesamtablauf
TODO: Phil: Muss noch auf deutsch, Mauszeiger raus und an finalen Ablauf anpassen

AbkÃ¼rzungsverzeichnis

Abbildungsverzeichnis

Literatur